Review

Last week

- Propositional Logic: Syntax
- Well-Formed Formulas (wffs)
- Induction and Recursion

Outline

- Recognizing Well-Formed Formulas
- Propositional Logic: Semantics
- Truth Tables
- Satisfiability and Tautologies
- Propositional Connectives and Boolean Functions
- Compactness
- Computability and Decidability

Sources:

Enderton, Sections: 1.2, 1.3, 1.5, 1.7.

N. J. Cutland. *Computability*.


Propositional Logic: Well-Formed Formulas

Recall our inductive definition of the set $W$ of well-formed formulas in propositional logic. Given the alphabet $\{(), \neg, \land, \lor, \rightarrow, \leftrightarrow, A_1, A_2, \ldots\}$,

- $U$ = the set of all expressions over the alphabet.
- $B$ = the set of expressions consisting of a single propositional symbol.
- $F$ = the set of formula-building operations:
  - $\mathcal{E}_\neg(\alpha) = (\neg \alpha)$
  - $\mathcal{E}_\land(\alpha, \beta) = (\alpha \land \beta)$
  - $\mathcal{E}_\lor(\alpha, \beta) = (\alpha \lor \beta)$
  - $\mathcal{E}_\rightarrow(\alpha, \beta) = (\alpha \rightarrow \beta)$
  - $\mathcal{E}_\leftrightarrow(\alpha, \beta) = (\alpha \leftrightarrow \beta)$
An Algorithm for Recognizing WFFs

Lemma

Let \( \alpha \) be a wff. Then exactly one of the following is true.

- \( \alpha \) is a propositional symbol.
- \( \alpha = (\neg \beta) \) where \( \beta \) is a wff.
- \( \alpha = (\beta \odot \gamma) \) where \( \odot \) is one of \( \{\land, \lor, \rightarrow, \leftrightarrow\} \), \( \beta \) is the first parentheses-balanced initial segment of the result of dropping the first \( ( \) from \( \alpha \), and \( \beta \) and \( \gamma \) are wffs.

How would you prove this?

Induction, of course!

An Algorithm for Recognizing WFFs

Input: expression \( \alpha \)
Output: true or false (indicating whether \( \alpha \) is a wff).

0. Begin with an initial construction tree \( T \) containing a single node labeled with \( \alpha \).
1. If all leaves of \( T \) are labeled with propositional symbols, return true.
2. Select a leaf labeled with an expression \( \alpha_1 \) which is not a propositional symbol.
3. If \( \alpha_1 \) does not begin with \( ( \) return false.
4. If \( \alpha_1 = (\neg \beta) \), then add a child to the leaf labeled by \( \alpha_1 \), label it with \( \beta \), and goto 1.
5. Scan \( \alpha_1 \) until first reaching \( ( \beta \), where \( \beta \) is a nonempty expression having the same number of left and right parentheses. If there is no such \( \beta \), return false.
6. If \( \alpha_1 = (\beta \odot \gamma) \) where \( \odot \) is one of \( \{\land, \lor, \rightarrow, \leftrightarrow\} \), then add two children to the leaf labeled by \( \alpha_1 \), label them with \( \beta \) and \( \gamma \), and goto 1.
7. Return false.

Termination

How do we prove termination of this algorithm?

We can show that the sum of the lengths of all the expressions labeling leaves decreases on each iteration of the loop.

Soundness

If the algorithm returns true when given input \( \alpha \), then \( \alpha \) is a wff.

The proof is by induction on the tree \( T \) generated by the algorithm from the leaves up to the root.

Completeness

If \( \alpha \) is a wff, then the algorithm will return true.

Proof using the induction principle for the set of wffs.

Notational Conventions

- Larger variety of propositional symbols: \( A, B, C, D, p, q, r \), etc.
- Outermost parentheses can be omitted: \( A \land B \) instead of \( (A \land B) \).
- Negation symbol binds stronger than binary connectives and its scope is as small as possible: \( \neg A \land B \) means \( ((\neg A) \land B) \).
- \( \{\land, \lor\} \) bind stronger than \( \{\rightarrow, \leftrightarrow\} \): \( A \land B \rightarrow \neg C \lor D \) is \( ((A \land B) \rightarrow ((\neg C) \lor D)) \).
- When one symbol is used repeatedly, grouping is to the right: \( A \land B \land C \) is \( (A \land (B \land C)) \).

Note that conventions are only unambiguous for wffs, not for arbitrary expressions.
Propositional Logic: Semantics

Intuitively, given a wff \( \alpha \) and a value (either \( T \) or \( F \)) for each propositional symbol in \( \alpha \), we should be able to determine the value of \( \alpha \).

How do we make this precise?

Let \( v \) be a function from \( B \) to \( \{ F, T \} \). We call this function a truth assignment.

Now, we define \( v \), a function from \( W \) to \( \{ F, T \} \) as follows (we compute with \( F \) and \( T \) as if they were \( 0 \) and \( 1 \) respectively).

- For each propositional symbol \( A_i \), \( v(A_i) = v(A_i) \).
- \( v(E\neg(\alpha)) = T - v(\alpha) \)
- \( v(E\land(\alpha, \beta)) = \min(v(\alpha), v(\beta)) \)
- \( v(E\lor(\alpha, \beta)) = \max(v(\alpha), v(\beta)) \)
- \( v(E\rightarrow(\alpha, \beta)) = \max(T - v(\alpha), v(\beta)) \)
- \( v(E\leftrightarrow(\alpha, \beta)) = T - |v(\alpha) - v(\beta)| \)

The recursion theorem and the unique readability theorem guarantee that \( v \) is well-defined.

Complex truth tables

Truth tables can also be used to calculate all possible values of \( v \) for a given wff:

We associate a column with each propositional symbol and a column with each propositional connective. There is a row for each possible truth assignment to the propositional connectives.

<table>
<thead>
<tr>
<th>( A_1 )</th>
<th>( A_2 )</th>
<th>( A_3 )</th>
<th>( (A_1 \lor (A_2 \land \neg A_3)) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

Definitions

If \( \alpha \) is a wff, then a truth assignment \( v \) satisfies \( \alpha \) if \( v(\alpha) = T \).

A wff \( \alpha \) is satisfiable if there exists some truth assignment \( v \) which satisfies \( \alpha \).

Suppose \( \Sigma \) is a set of wffs. Then \( \Sigma \) tautologically implies \( \alpha \), \( \Sigma \models \alpha \), if every truth assignment which satisfies each formula in \( \Sigma \) also satisfies \( \alpha \).

Particular cases:
- If \( \emptyset \models \alpha \), then we say \( \alpha \) is a tautology or \( \alpha \) is valid and write \( \models \alpha \).
- If \( \Sigma \) is unsatisfiable, then \( \Sigma \models \alpha \) for every wff \( \alpha \).
- If \( \alpha \models \beta \) (shorthand for \( \{ \alpha \} \models \beta \)) and \( \beta \models \alpha \), then \( \alpha \) and \( \beta \) are tautologically equivalent.
- \( \Sigma \models \alpha \) if and only if \( \land(\Sigma) \rightarrow \alpha \) is valid.

Truth Tables

There are other ways to present the semantics which are less formal but perhaps more intuitive.

<table>
<thead>
<tr>
<th>( \alpha )</th>
<th>( \neg \alpha )</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( \alpha )</th>
<th>( \beta )</th>
<th>( \alpha \land \beta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( \alpha )</th>
<th>( \beta )</th>
<th>( \alpha \lor \beta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( \alpha )</th>
<th>( \beta )</th>
<th>( \alpha \rightarrow \beta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( \alpha )</th>
<th>( \beta )</th>
<th>( \alpha \leftrightarrow \beta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>
Examples

• \((A \lor B) \land (\neg A \lor \neg B)\) is satisfiable, but not valid.
• \((A \lor B) \land (\neg A \lor \neg B) \land (A \leftrightarrow B)\) is unsatisfiable.
• \(\{A, A \rightarrow B\} \models B\) \((A \land (A \rightarrow B) \land (\neg B))\)
• \(\{A, \neg A\} \models (A \land \neg A) \land (\neg (A \land \neg A))\)
• \(\neg (A \land B)\) is tautologically equivalent to \(\neg A \lor \neg B\)
  \((- (A \land B) \leftrightarrow (\neg A \lor \neg B))\)

Suppose you had an algorithm \(\text{SAT}\) which would take a \(wff\) \(\alpha\) as input and return \(\text{true}\) if \(\alpha\) is satisfiable and \(\text{false}\) otherwise. How would you use this algorithm to verify each of the claims made above?

Determining Satisfiability using Truth Tables

An Algorithm for Satisfiability
To check whether \(\alpha\) is satisfiable, form the truth table for \(\alpha\). If there is a row in which \(T\) appears as the value for \(\alpha\), then \(\alpha\) is satisfiable. Otherwise, \(\alpha\) is unsatisfiable.

An Algorithm for Tautological Implication
To check whether \(\{\alpha_1, \ldots, \alpha_k\} \models \beta\), check the satisfiability of \((\alpha_1 \land \ldots \land \alpha_k) \land (\neg \beta)\). If it is unsatisfiable, then \(\{\alpha_1, \ldots, \alpha_k\} \models \beta\), otherwise \(\{\alpha_1, \ldots, \alpha_k\} \not\models \beta\).
Determining Satisfiability using Truth Tables

What is the complexity of this algorithm?

$2^n$ where $n$ is the number of propositional symbols.

Can you think of a way to speed up these algorithms?

In an upcoming lecture, we will discuss some of the applications and best-known techniques for the SAT algorithm.

Some tautologies

**Associative and Commutative laws for $\land$, $\lor$, $\leftrightarrow$**

**Distributive Laws**

- $(A \land (B \lor C)) \leftrightarrow ((A \land B) \lor (A \land C))$.
- $(A \lor (B \land C)) \leftrightarrow ((A \lor B) \land (A \lor C))$.

**Negation**

- $\neg\neg A \leftrightarrow A$
- $\neg(A \to B) \leftrightarrow (A \land \neg B)$
- $\neg(A \leftrightarrow B) \leftrightarrow ((A \land \neg B) \lor (\neg A \land B))$.

**De Morgan's Laws**

- $\neg(A \land B) \leftrightarrow (\neg A \lor \neg B)$
- $\neg(A \lor B) \leftrightarrow (\neg A \land \neg B)$

**Propositional Connectives**

We have five connectives: $\neg$, $\land$, $\lor$, $\to$, $\leftrightarrow$. Would we gain anything by having more? Would we lose anything by having fewer?

**Example: Ternary Majority Connective #**

$E_#(\alpha, \beta, \gamma) = (#\alpha\beta\gamma)$

$\overline{v}(#\alpha\beta\gamma) = \mathbf{T}$ iff the majority of $\overline{v}(\alpha)$, $\overline{v}(\beta)$, and $\overline{v}(\gamma)$ are $\mathbf{T}$.

What does this new connective do for us?

**Claim:** The extended language obtained by allowing this new symbol has the same expressive power as the original language.

How do we show this formally?
Boolean Functions

For $k \geq 0$, a $k$-place Boolean function is a function from $\{F, T\}^k$ to $\{F, T\}$. A Boolean function then is anything which is a $k$-place Boolean function for some $k$.

Each wff $\alpha$ determines a corresponding Boolean function $B_\alpha$. For example, if $\alpha = A_1 \wedge A_2$, then $B_\alpha$ is a 2-place Boolean function whose value is given by the following table.

<table>
<thead>
<tr>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$B_\alpha(X_1, X_2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

Examples

- $I^n = B^n_A$
- $N = B^1_{A_1}$
- $K = B^2_{A_1 \wedge A_2}$
- $A = B^2_{A_1 \vee A_2}$
- $C = B^2_{A_1 \rightarrow A_2}$
- $E = B^2_{A_1 \leftrightarrow A_2}$

From these functions, we can construct others by composition.

$$B^2_{\neg A_1 \vee \neg A_2}(X_1, X_2) = A(N(I^n_1(X_1, X_2)), N(I^n_2(X_1, X_2)))$$

Claim: Every Boolean function can be obtained as a composition of $I$, $N$, $K$, $A$, $C$, and $E$.

We will explain why this is true shortly.

Realizing Boolean Functions

In general, suppose that $\alpha$ is a wff whose propositional symbols are included in $A_1, \ldots, A_n$. We define an $n$-place Boolean function $B^n_\alpha$, the Boolean function realized by $\alpha$ as

$$B^n_\alpha(X_1, \ldots, X_n) = \text{the truth value given to } \alpha \text{ when } A_1, \ldots, A_n \text{ are given the values } X_1, \ldots, X_n.$$

In other words,

$$B^n_\alpha(X_1, \ldots, X_n) = v(\alpha) \text{ where } v(A_i) = X_i.$$

Note that the function $B^n_\alpha$ is determined by both the formula $\alpha$ and the choice of $n$. In particular, $\alpha$ does not need to include all the symbols in $A_1, \ldots, A_n$.

Formulas and the Boolean Functions they Realize

Theorem

Let $\alpha$ and $\beta$ be wffs whose sentence symbols are among $A_1, \ldots, A_n$.

(a) $\alpha \models \beta$ iff $B^n_\alpha(X) \leq B^n_\beta(X)$ for all $X \in \{F, T\}^n$.

(b) $\alpha$ is tautologically equivalent to $\beta$ iff $B^n_\alpha = B^n_\beta$.

(c) $\models \beta$ iff the range of $B^n_\beta$ is $\{T\}$.

Proof

(a)

$$\alpha \models \beta \iff \text{for every truth assignment satisfying } \alpha \text{ also satisfies } \beta$$

(b) Follows from (a) and $X = Y$ iff $X \leq Y$ and $Y \leq X$.

(c) Follows from (a) and definition of tautology.

By shifting our focus from formulas to Boolean functions, tautologically equivalent wffs are identified.
Completeness of Propositional Connectives

**Theorem**

Let $G$ be an $n$-place Boolean function, $n \geq 1$. There exists a **wff** $\alpha$ such that $G = B^n_\alpha$, i.e., such that $\alpha$ realizes the function $G$.

**Proof**

If the range of $G$ is just $\{F\}$, then let $\alpha = A_1 \land \neg A_1$. Clearly, $B^n_\alpha = G$.

Otherwise, $G = T$ somewhere. Suppose there are $k$ points where $G = T$:

\[
G(X_{11}, X_{12}, \ldots, X_{1n}) = T \\
G(X_{21}, X_{22}, \ldots, X_{2n}) = T \\
\vdots \\
G(X_{k1}, X_{k2}, \ldots, X_{kn}) = T
\]

Let $\beta_{ij} = \begin{cases} A_j & \text{if } X_{ij} = T \\ \neg A_j & \text{if } X_{ij} = F \end{cases}$

$\gamma_i = \beta_{i1} \land \ldots \land \beta_{in}$

$\alpha = \gamma_1 \lor \gamma_2 \lor \ldots \lor \gamma_k$

Then $\alpha$ realizes $G$.


discjunctive normal form (DNF). A formula is in DNF if it is a disjunction of formulas, each of which is a conjunction of **literals**, where a literal is either a propositional symbol or its negation.

Thus, a corollary is that for every **wff**, there exists a tautologically equivalent **wff** in disjunctive normal form.

Completeness of Propositional Connectives

**Example**

Let $G$ be a 3-place Boolean function defined as follows:

$G(F, F, F) = F$

$G(F, F, T) = T$

$G(F, T, F) = T$

$G(F, T, T) = F$

$G(T, F, F) = T$

$G(T, F, T) = F$

$G(T, T, F) = F$

$G(T, T, T) = T$

There are four points at which $G$ is true, so a DNF formula which realizes $G$ is

\[
(\neg A_1 \land \neg A_2 \land A_3) \lor (\neg A_1 \land A_2 \land \neg A_3) \lor (A_1 \land \neg A_2 \land \neg A_3) \lor (A_1 \land A_2 \land A_3).
\]

Note that another formula which realizes $G$ is $A_1 \leftrightarrow A_2 \leftrightarrow A_3$. Thus, adding additional connectives to a complete set may allow a function to be realized more concisely.

Completeness of Propositional Connectives

Recall our definition of some basic Boolean functions:

- $I^n_1 = B^n_{A_1}$
- $N = B^1_{A_1}$
- $K = B^3_{A_1 \land A_2}$
- $A = B^2_{A_1 \lor A_2}$

Given that $\{\neg, \land, \lor\}$ is complete, it is not hard to see that any Boolean function can be constructed using only the Boolean functions $I$, $N$, $K$, and $A$.

In fact, we can do better. It turns out that $\{\neg, \land\}$ and $\{\neg, \lor\}$ are complete as well.

Why?

$\alpha \lor \beta \leftrightarrow \neg (\neg \alpha \land \neg \beta)$

$\alpha \land \beta \leftrightarrow \neg (\neg \alpha \lor \neg \beta)$

Using these identities, the completeness can be easily proved by induction.
Incompleteness of Connectives

To prove that some set of connectives is incomplete, we find a property that is true of all wffs built using those connectives, but that is not true for some Boolean function.

Example

\{\land, \rightarrow\} is not complete.

Proof

Let \( \alpha \) be a wff which uses only these connectives, and let \( v \) be a truth assignment such that \( v(A_i) = T \) for all \( A_i \). We prove by induction that \( v(\alpha) = T \).

Base Case

\( v(A_i) = v(A_i) = T \).

Inductive Case

\[ v(\beta \land \gamma) = \max(v(\beta), v(\gamma)) = \max(T, T) = T \]
\[ v(\beta \rightarrow \gamma) = \max(T - v(\alpha), v(\beta)) = \max(F, T) = T \]

Thus, \( v(\alpha) = T \) for all wffs \( \alpha \) built from \{\land, \rightarrow\}. But \( v(\neg A_1) = F \), so there is no such formula tautologically equivalent to \( \neg A_1 \). \( \square \)

Other Propositional Connectives

For each \( n \), there are \( 2^{2^n} \) different \( n \)-place Boolean functions \( B(X_1, \ldots, X_n) \).

Why?

There are \( 2^n \) different input points and 2 possible output values for each input point. \( 2^{2^n} \) is also the number of possible \( n \)-ary propositional connectives.

0-ary connectives

There are two 0-place Boolean functions: the constants \( F \) and \( T \). We can construct corresponding 0-ary connectives \( \bot \) and \( \top \) with the meaning that \( v(\bot) = F \) and \( v(\top) = T \) regardless of the truth assignment \( v \).

Unary connectives

There are four 1-place functions, but these include the two constant functions mentioned above and the identity function. Thus the only additional connective of interest is negation: \( \neg \).

Binary connectives

There are sixteen 2-place Boolean functions. They are cataloged in the following table. Note that the first six correspond to 0-ary and unary connectives.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Equivalent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \bot )</td>
<td>constant ( F )</td>
<td></td>
</tr>
<tr>
<td>( \top )</td>
<td>constant ( T )</td>
<td></td>
</tr>
<tr>
<td>( A )</td>
<td>projection of first argument</td>
<td></td>
</tr>
<tr>
<td>( B )</td>
<td>projection of second argument</td>
<td></td>
</tr>
<tr>
<td>( \neg A )</td>
<td>negation of first argument</td>
<td></td>
</tr>
<tr>
<td>( \neg B )</td>
<td>negation of second argument</td>
<td></td>
</tr>
<tr>
<td>( \land )</td>
<td>( A \land B )</td>
<td>and</td>
</tr>
<tr>
<td>( \lor )</td>
<td>( A \lor B )</td>
<td>or</td>
</tr>
<tr>
<td>( \rightarrow )</td>
<td>( A \rightarrow B )</td>
<td>conditional</td>
</tr>
<tr>
<td>( \leftrightarrow )</td>
<td>( A \leftrightarrow B )</td>
<td>bi-conditional</td>
</tr>
<tr>
<td>( \leftarrow )</td>
<td>( B \rightarrow A )</td>
<td>reverse conditional</td>
</tr>
<tr>
<td>( \oplus )</td>
<td>( (A \land \neg B) \lor (\neg A \land B) )</td>
<td>exclusive or</td>
</tr>
<tr>
<td>( \downarrow )</td>
<td>( \neg(A \lor B) )</td>
<td>nor (or Nicod stroke)</td>
</tr>
<tr>
<td>( \mid )</td>
<td>( \neg(A \land B) )</td>
<td>nand (or Sheffer stroke)</td>
</tr>
<tr>
<td>( &lt; )</td>
<td>( \neg A \land B )</td>
<td>less than</td>
</tr>
<tr>
<td>( &gt; )</td>
<td>( A \land \neg B )</td>
<td>greater than</td>
</tr>
</tbody>
</table>

Compactness

Recall that a wff \( \alpha \) is satisfiable if there exists a truth assignment \( v \) such that \( v(\alpha) = T \).

A set \( \Sigma \) of wffs is satisfiable if there exists a truth assignment \( v \) such that \( v(\alpha) = T \) for each \( \alpha \in \Sigma \).

A set \( \Sigma \) is finitely satisfiable iff every finite subset of \( \Sigma \) is satisfiable.

Compactness Theorem

A set of wffs is satisfiable if it is finitely satisfiable.

Proof

The only if direction is trivial since any subset of a satisfiable set is clearly satisfiable.

To prove the other direction, assume that \( \Sigma \) is a set which is finitely satisfiable. We must show that \( \Sigma \) is satisfiable.
Compactness

Let $\Sigma$ be finitely satisfiable. We extend $\Sigma$ to form a maximal finitely satisfiable set $\Delta$ as follows.

Let $\alpha_1, \ldots, \alpha_n, \ldots$ be a fixed enumeration of all wffs.

Why is this possible? The set of all sequences of a countable set is countable.

Then, let $\Delta_0 = \Sigma$, $\Delta_{n+1} = \begin{cases} \Delta_n \cup \{\alpha_{n+1}\} & \text{if this is finitely satisfiable,} \\ \Delta_n \cup \{\neg \alpha_{n+1}\} & \text{otherwise.} \end{cases}$

It is not hard to show that each $\Delta_n$ is finitely satisfiable.

Let $\Delta = \bigcup_n \Delta_n$. It is then clear that

1. $\Sigma \subseteq \Delta$
2. $\alpha \in \Delta$ or $\neg \alpha \in \Delta$ for any wff $\alpha$, and
3. $\Delta$ is finitely satisfiable.

Compactness

Corollary

If $\Sigma \models \alpha$ then there is a finite $\Sigma_0 \subseteq \Sigma$ such that $\Sigma_0 \models \alpha$.

Proof

Suppose that $\Sigma_0 \not\models \alpha$ for every finite $\Sigma_0 \subseteq \Sigma$.

Then, $\Sigma_0 \cup \{\neg \alpha\}$ is satisfiable for every finite $\Sigma_0 \subseteq \Sigma$.

So, by compactness, $\Sigma \cup \{\neg \alpha\}$ is satisfiable which contradicts the fact that $\Sigma \models \alpha$.

Compactness

Now we show that $\Delta$ is satisfiable (and thus $\Sigma \subseteq \Delta$ is also satisfiable).

Define a truth assignment $v$ as follows. For each propositional symbol $A_i$,

$v(A_i) = T$ iff $A_i \in \Delta$.

We claim that for any wff $\alpha$, $v$ satisfies $\alpha$ iff $\alpha \in \Delta$. The proof is by induction on well-formed formulas.

Base Case

Follows directly from the definition of $v$.

Induction Case

We will just consider one case. Suppose $\alpha = \beta \land \gamma$. Then $v(\alpha) = T$ iff both $v(\beta) = T$ and $v(\gamma) = T$ iff both $\beta \in \Delta$ and $\gamma \in \Delta$.

Now, if both $\beta$ and $\gamma$ are in $\Delta$, then since $\{\beta, \gamma, \neg \alpha\}$ is not satisfiable, we must have $\alpha \in \Delta$.

Similarly, if one of $\beta$ or $\gamma$ is not in $\Delta$, then its negation must be in $\Delta$, so $\alpha \not\in \Delta$. □

Computability

The important notion of computability relies on a formal model of computation.

Many formal models have been proposed:

1. General recursive functions defined by means of an equation calculus (Gödel-Herbrand-Kleene)
2. $\lambda$-definable functions (Church)
3. $\mu$-recursive functions and partial recursive functions (Gödel-Kleene)
4. Functions computable by finite machines known as Turing machines (Turing)
5. Functions defined from canonical deduction systems (Post)
6. Functions given by certain algorithms over a finite alphabet (Markov)
7. Universal Register Machine-computable functions (Shepherdson-Sturgis)

Fundamental Result

All of these (and many other) models of computation are equivalent. That is, they give rise to the same class of functions.
Computability and Decidability

All of these models are equivalent to what can be achieved by a computer with any standard programming language, given arbitrary (but finite) time and memory.

Church’s Thesis

A notion known as Church’s thesis states that all models of computation are either equivalent to or less powerful than those just described.

We will accept Church’s thesis and thus define a function to be computable if we can describe precisely (using any model of computation) how to compute it. Such a description will be called an effective procedure.

Decidability

Given a universal set \( U \), a set \( S \subseteq U \) is decidable if there exists a computable function \( f : U \rightarrow \{ \text{F}, \text{T} \} \) such that \( f(x) = \text{T} \) iff \( x \in S \).

Decidability of \( W \)

Earlier, we presented an algorithm which, given any expression \( \alpha \) determines whether the expression is well-formed. Thus, the set \( W \) of well-formed formulas is decidable.

Semi-Decidability

Suppose we wish to determine whether \( \Sigma \models \alpha \) where \( \Sigma \) is infinite. In general, this is not decidable.

But we can obtain a weaker result:

A set \( A \) is semi-decidable (or effectively enumerable) if there is an effective procedure which lists, in some order, every member of \( A \).

Note that if \( A \) is infinite, then the procedure will never finish, but every member of \( A \) must appear in the list after some finite amount of time.

Theorem

A set \( A \) of expressions is effectively enumerable iff there is an effective procedure which, given any expression \( \alpha \), produces the answer “yes” iff \( \alpha \in A \).

Proof

If \( A \) is effectively enumerable, then we simply enumerate its members and check each one to see if it is equivalent to \( \alpha \). If it is, we return “yes” and stop. Otherwise, we keep going. Thus, if \( \alpha \in A \), the procedure produces “yes”. If \( \alpha \notin A \), the procedure runs forever.

Decidability

Some decidable sets

- For a given finite set of well-formed \( \Sigma \), the set of all tautological consequences of \( \Sigma \) (i.e., \( \{ \alpha \mid \Sigma \models \alpha \} \)) is decidable.
  - The truth table algorithm given earlier decides \( \Sigma \models \alpha \).
- The set of tautologies is decidable.
  - The set of tautologies is just the set of tautological consequences of the empty set.

Existence of undecidable sets

A simple argument shows the existence of undecidable sets of expressions: an algorithm is completely determined by its finite description. Thus, there are only countably many effective procedures. But there are uncountably many sets of expressions.

Why?

The set of expressions is countably infinite. Therefore, its power set is uncountable.

Proof, continued

On the other hand, suppose that we have an effective procedure \( P \) which produces “yes” iff \( \alpha \in A \). To produce an enumeration of \( A \), we proceed as follows. First enumerate all expressions:

\( \varepsilon_1, \varepsilon_2, \varepsilon_3, \ldots \)

Then proceed as follows.

- Break the procedure \( P \) into a finite number of “steps”.
- Run \( P \) on \( \varepsilon_1 \) for 1 step.
- Run \( P \) on \( \varepsilon_1 \) for 2 steps, and then run \( P \) on \( \varepsilon_2 \) for 2 steps.
  - \( \ldots \)
- Run \( P \) on each of \( \varepsilon_1, \ldots, \varepsilon_n \) for \( n \) steps each
  - \( \ldots \)

If at any time, the procedure \( P \) produces “yes”, then we list the expression which produced “yes” and continue.

This procedure will eventually enumerate all members of \( A \).
Semi-Decidability

Theorem
A set is decidable iff both it and its complement (with respect to a given universal set) are effectively enumerable.

Proof
Alternate between running the procedure for the set and the procedure for its complement. One of them will eventually produce “yes”.

Properties of decidable and semi-decidable sets
Decidable sets are closed under union, intersection, and complement.

Semi-decidable sets are closed under union and intersection.

Semi-Decidability

Theorem
If \( \Sigma \) is an effectively enumerable set of wffs, then the set of tautological consequences of \( \Sigma \) is effectively enumerable.

Proof
Consider an enumeration of the elements of \( \Sigma \):
\[ \sigma_1, \sigma_2, \sigma_3, \ldots \]

By the compactness theorem, \( \Sigma \models \alpha \) iff \( \{\sigma_1, \ldots, \sigma_n\} \models \alpha \) for some \( n \).

Hence, it is sufficient to successively test:

\[ \emptyset \models \alpha \]
\[ \{\sigma_1\} \models \alpha \]
\[ \{\sigma_1, \sigma_2\} \models \alpha \]
\[ \ldots \]

If any of these conditions is met (each of which is decidable), the answer is “yes”.

Semi-Decidability

Theorem (continued)
This demonstrates that there is an effective procedure that, given any wff \( \alpha \), will output “yes” iff \( \alpha \) is a tautological consequence of \( \Sigma \).

Thus, the set of tautological consequences of \( \Sigma \) is effectively enumerable. \( \square \)